CMPT 431: Distributed Systems (Fall 2019)

Assignment 2 - Report

Name	Muhammad Arslan
SFU ID	301296874

Instructions:

- This report is worth 30 points.
- Answer in the space provided.
  Answers spanning beyond 3 lines (11pt font) will lose points.
- Input graphs used are available at the following location.
  - live-journal graph (LJ graph): `/scratch/assignment2/input_graphs/lj`
  - RMAT graph: `/scratch/assignment2/input_graphs/rmat`
- All the experiments are conducted with 4 workers.
- All the times are in seconds.

---

1. [4 points] Run Triangle Counting with `--strategy=1` on the LJ graph and the RMAT graph. Update the thread statistics in the tables below. What is your observation on the difference in time taken by each thread for RMAT and that for LJ? Why does this happen?

Answer:

Although there is a huge difference between the size of the graph, RMAT is much bigger than LJ, but it takes a lot less time. This is because vertices is RMAT has lower out degree and hence forming less triangles.So the loop that is run out degree times is cut short and reducing the total time.vim

Triangle Counting on LJ: Total time = 56.49615 seconds.

thread_id	num_vertices	num_edges	triangle_count	time_taken
0	1211892	42920131	357657286	56.49562
1	1211892	15515692	217447326	12.82936
2	1211892	7141449	86161667	3.49256
3	1211895	3416501	46058470	0.95605

Triangle Counting on RMAT: Total time = 4.35863 seconds.

thread_id	num_vertices	num_edges	triangle_count	time_taken
0	6249999	12650749	7	4.22376
1	6249999	12546666	5	4.35818
2	6249999	12399926	6	3.23894
3	6250002	12402659	9	3.24919

2. [3 points] Run Triangle Counting with `--strategy=2` on LJ graph. Update the thread statistics in the table below. Partitioning time is the time spent on task decomposition as required by `--strategy=2`. What is your observation on the difference in time taken by each thread, and the difference in num_edges for each thread? Are they correlated (yes/no)? Why?

Answer:

The difference in time taken by each thread is because not all edges form equal number of triangles. The number of edges were equally divided as suggested by strategy and number of edges and time_taken for each thread are not correlated because the equal num of edges does not equal the time taken

Triangle Counting on LJ: Partitioning time = 2.13819 seconds. Total time = 30.80949 seconds.

thread_id	num_vertices	num_edges	triangle_count	time_taken
0	0	17248443	144441858	28.14378
1	0	17248443	152103585	21.93599
2	0	17248443	225182666	16.10273
3	0	17248444	185596640	8.67537

3. [1 point] Run Triangle Counting with `--strategy=3` on LJ graph. Update the thread statistics in the table below.

Triangle Counting on LJ: Total time = 21.49212 ___ seconds.

thread_id	num_vertices	num_edges	triangle_count	time_taken
0	1213044	17220413	176722047	21.49126
1	1225854	17391859	177985487	21.49122
2	1205828	17235371	176706809	21.49114
3	1202845	17146130	175910406	21.49111

4. [3 points] Run PageRank with `--strategy=1` on LJ graph. Update the thread statistics in the table below. What is your observation on the difference in time taken by each thread, and the difference in num_edges for each thread? Is the work uniformly distributed across threads (yes/no)? Why?

Answer:

all threads almost took same time. This is because in the parallel implementation each thread waits for other threads to finish. The number of edges vary by a significant number. Work is not distributed uniformly. The work depends on processing number edges and number of vertices processed, which is not same for easch thread.

PageRank on LJ: Total time = 63.12975 ___ seconds.

thread_id	num_vertices	num_edges	time_taken
0	24237840	858402620	63.12595
1	24237840	310313840	63.12825
2	24237840	142828980	63.12606
3	24237900	68330020	63.12686

5. [3 points] Run PageRank with `--strategy=1` on LJ graph. Obtain the cumulative time spent by each thread on **`barrier1`** and **`barrier2`** (refer pagerank pseudocode for program 4 on assignment webpage) and update the table below. What is your observation on the difference in barrier1_time for each thread and the difference in num_edges for each thread? Are they correlated (yes/no)? Why?

Answer:

thread 0 has the most amount of work by looking at the number of edges. It takes the longest time to finish, having shortest barrier1_time. Other threads that finish earlier has to wait very long for thread 1 a shown in the table. Thread3 waiting for 56 seconds sitting idle

PageRank on LJ: Total time = 63.12975 seconds.

thread_id	num_vertices	num_edges	barrier1_time	barrier2_time	time_taken
0	24237840	858402620	0.000635	0.067275	63.12595
1	24237840	310313840	34.07638	0.00806	63.12825
2	24237840	142828980	49.76312	0.03842	63.12606
3	24237900	68330020	56.84230	0.01536	63.12686

6. [3 points] Run PageRank with `--strategy=2` on the LJ graph. Update the thread statistics in the table below. Update the time taken for task decomposition as required by `--strategy=2`. What is your observation on barrier2_time compared to the barrier2_time in question 5 above? Why are they same/different?

Answer:

the barrier2_time is only related to vertices. In question5 we have equal number of vertices distributed among threads so barrier time didn't differ by much. In strategy 2, the idea was to divide work giving approximately equal number of edges so vertices were different for all, hence resulting a big difference in the barrier2_time.

PageRank on LJ: Total time = 35.19299 seconds. Partitioning time = 0.01135 seconds.

thread_id	num_vertices	num_edges	barrier1_time	barrier2_time	time_taken
0	6510800	344968340	0.485705	3.236804	35.03111
1	10210900	344969340	0.43989	2.99944	35.04203
2	18080820	344968580	0.98633	2.55260	35.06402

3	62148900	344969200	3.05308	0.00062	35.19106

7. [3 points] Run PageRank with `--strategy=3` on LJ graph. Update the thread statistics in the table below. What is your observation on barrier times compared to the barrier times in question 6 above? What is your observation on the time taken by each thread compared to time taken by each thread in question 6 above? Why are they same/different?

Answer:

the barrier time for strategy 3 are lower relative to strategy 2. This is because barrier are dependent on how the work is distributed

In latter case, each thread only get work assigned if it is free. Pre-assigned vertices can be a problem because some vertices need more work than other. In strategy 3, we can say that, number of vertices/edges are not equally distributed but the work is equally distributed.

PageRank on LJ: Total time = 100.76003 seconds. Partitioning time = 66.11586 seconds.

thread_id	num_vertices	num_edges	barrier1_time	barrier2_time	time_taken
0	24279250	339250283	0.001373	0.000931	100.74863
1	24299856	346389335	0.00152	0.00088	100.74708
2	24178627	346803134	0.00144	0.00085	100.74543
3	24193687	347432708	0.00146	0.00088	100.70514

8. [3 points] Run PageRank with `--strategy=3` on LJ graph. Obtain the total time spent by each thread in `getNextVertexToBeProcessed()` and update the table below. What is your observation on the time taken by `getNextVertexToBeProcessed()`? Why is it high/low?

Answer:

It is very high, this is because a thread tries to get a new vertex after each vertex is processed. this overhead to get vertex so many times adds to a larger value

PageRank on LJ: Total time = 100.76003 seconds.

thread_id	num_vertices	num_edges	barrier1_time	barrier2_time	getNextVertex_time	time_taken
0	24279250	339250283	0.001373	0.000931	16.604629	100.74863
1	24299856	346389335	0.00152	0.00088	16.53037	100.74708

2	24178627	346803134	0.00144	0.00085	16.46120	100.74543
3	24193687	347432708	0.00146	0.00088	16.51966	100.70514

9. [3 points] Run PageRank with `--strategy=3` on LJ graph with `--granularity=2000`. Update the thread statistics in the table below. What is your observation on the time taken by `getNextVertexToBeProcessed()`? Why is it high/low?

Answer:
It is low because, a thread gets n(20000) number of vertices at a time. which reduces the total overhead of calling the get vertex function less frequently.

PageRank on LJ: Granularity = 2000. Total time = 36.62069 seconds. Partitioning time = 0.03122 seconds.

thread_id	num_vertices	num_edges	barrier1_time	barrier2_time	getNextVertex_time	time_taken
0	23914997	341030205	0.002901	0.003485	0.007871	36.60778
1	24361426	339166786	0.00209	0.01157	0.00759	36.59744
2	24111571	349895746	0.00246	0.01164	0.00770	36.59279
3	24563426	349782723	0.00281	0.00935	0.00806	36.59440

10. [4 points] While --strategy=3 with --granularity=2000 performs best across all of our parallel PageRank attempts, it doesn't give much performance benefits over our serial program (might give worse performance on certain inputs). Why is this the case? How can the parallel solution be improved further to gain more performance benefits over serial PageRank?

Answer:
this is the case because the overhead of distributing the work and also the work that is divided is not equal between the threads which makes thread idle for few time units.